REMARKS

Applicants have thoroughly considered the Examiner's remarks and the application has been amended to more clearly define the invention. Claims 1-38 are presented in the application for further examination. Claims 1-6, 8, 10, 12, 15-19 and 21-32 have been amended by this Amendment B. Claims 39-44 are canceled. Reconsideration of the application claims as amended and in view of the following remarks is respectfully requested. The following remarks will follow the sequence of the Office action. The Arabic numerals beginning each paragraph correspond to the numbered paragraphs of the Office action.

Claim Rejections - 35 USC §101

1. Claims 23-26 and 28-32 have been amended to delete the reference to P so that the objection to these claims should be withdrawn.

2. Claim 27 have been amended to provide antecedent basis for pixel so that the objection to claims 28-32 should be withdrawn.

3. Claims 39-44 have been canceled as duplicates.

4. Claim 21 has been amended to recite a tangible computer readable storage medium so that the rejection of claims 21-26 under 35 USC §101 should be withdrawn.

5. Claim 27 has been amended to recite a method of modifying pixel values of a digital video image so that the rejection of this claim based on 35 USC §101 should be withdrawn.

Claim Rejections - 35 USC §102

1.-2. Claims 1, 15-19 and 27 stand rejected under 35 USC §102(e) as being anticipated by Herf (6925210). The Examiner argues that:

blurred value. Herf discloses in lines 33-50 of column 1 the use of a box filter to blur a digital image pixel by pixel. In this method the box filter is used to take the average of the selected P pixels surrounding the center (particular) pixel or what is chosen to be the center pixel and then input this average into the particular pixel. This is done for each desired pixel in the digital image and results in a blurred version of the original image.

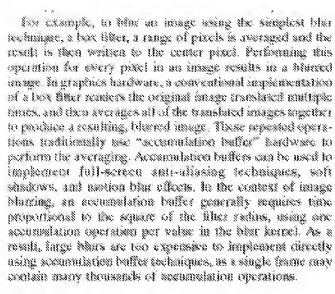
Independent claims 1, 15 and 27 have been amended to recite that "the selected pixels are **different** from the particular pixel" in combination with "determining a blurred value as a function of **only** the values of the selected pixels." This combination is not taught by Herf, which suggests a box filter. In general, a box filter includes the selected pixel in the blurred value calculation. See, for example the following references regarding box filters, copies of which are attached:

1.http://www-csl.csres.utexas.edu/users/billmark/teach/cs384g-05-fall/projects/impressionist/imageproc.html

2.http://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm

3.http://wiki.blender.org/index.php/Manual/Oversampling (Antialiasing)

Further, Herf teaches away from the invention as column 1, lines 33-50:



for example, to blur an image using the simplest blur technique, a box filter, a range of pixels is averaged and the result is then written to the center pixel. Performing this operation for every pixel in an image results in a blurred image. In graphics hardware, a conventional implementation of a box filter renders the original image translated multiple times, and then averages all of the translated images together to produce a resulting, blurred image. These repeated operations traditionally use "accumulation buffer" hardware to perform the averaging. Accumulation buffers can be used to implement full-screen anti-aliasing techniques, soft shadows, and motion blur effects. In the context of image blurring, an accumulation buffer generally requires time proportional to the square of the filter radius, using one accumulation operation per value in the blur kernel. As a result, large blurs are too expensive to implement directly using accumulation buffer techniques, as a single frame may contain many thousands of accumulation operations.

Furthermore, Herf is consistent with the attached references and indicates that the box filter is a range of pixels which averaged. The range necessarily includes the "center pixel," which according to Herf, is the pixel which is replaced after averaging. Thus, Herf teaches away from amended claims 1, 15 and 27 and the rejection based on §102 should be withdrawn.

Claims 16-19 are patentable based on their dependency from claim 15. In addition, claims 16-19 have been amended to further limit claim 15 so that the rejection based on §102 should be withdrawn.

Claim Rejections - 35 USC §103

3.-5. Claims 1-13, 21-25, 27-31, 33-37 and 39-43 stand rejected under 35 USC §103(a) as being unpatentable over Herf in view of Koike (5408338). The Examiner cites Kioke as teaching selecting the number of pixels:

based on the average of a plurality of pixels. However, Herf does not disclose a particular number of pixels used in this operation. Koike discloses in Figure 2, lines 20-44 of column 1, and in lines 1-51 of column 4 an image processing unit and method in which smoothing is performed. The smoothing operation as described by Koike is an evenly weighted average that is identical to the blur operation by this application. This smoothing (blurring) involves selecting a particular pixel and $P=2^N$ (with $N=$ 3 or 4) other pixels and blurring the particular pixel based on the average of the pixels. However, Koike does not exclude the particular pixel from the

However, the Examiner admits that Kioke does not exclude the particular pixel:

on the average of the pixels. However, Koike does not exclude the particular pixel from the calculation of the blurred value as can be seen in the equation provided and instead excludes pixel H. The reason a pixel is excluded from the calculation is that there is a benefit in digitally processing in powers of 2 and thus operating with 8 pixels is desired. Therefore arbitrarily removing a pixel such as pixel H is a benefit and the choice of H is merely an example used to illustrate Koike's process. It would have been obvious to one of ordinary skill in the art to exclude the particular pixel X or any of the other particular pixels to achieve the desired number of pixels in the selected set as shown by Koike to achieve the benefits of digitally processing powers of 2. Furthermore, it would have been obvious to one of ordinary skill in the art to combine the teachings of Koike with Herf since they both relate to the smoothing (blurring) of digital images to achieve improved computational efficiency.

The Examiner argues that it would have been obvious to exclude the particular pixel. However, Herf teaches to the contrary, as noted above. Therefore, the combination of Herf and Koike are consistent with each other and teach that the particular pixel is included.

Applicants note that the Examiner has not cited any art which teaches excluding the particular pixel, as recited by the amended claims. "[T]he question is whether there is something in the prior art as a whole to suggest the desirability, and thus the obviousness, of making the combination." *Lindemann MaschinenFabrick GMBH v. American Hoist and Derrick Company*, 730 F.2d 1452, 1462; 221 U.S.P.Q. 481, 488 (Fed. Cir. 1984). As has been shown, the non-analogous teachings of the prior art relate to including the particular pixel in the combination. Therefore, nothing in the cited references suggest their combination to exclude the particular pixel. Indeed, the Examiner failed to cite any basis whatsoever for combining these references. In fact, the Examiner's rejection provides a text book example of impermissible hindsight analysis -- the Examiner used the invention as defined by the claims as a guide in order to reject the claims. See In re Oetiker, 977 F.2d at 1447; 24 U.S.P.Q.2d at 1446 ("There must be some reason, suggestion, or motivation found in the prior art whereby a person of ordinary skill in the field of the invention would make the combination. That knowledge can not come from the applicant's invention itself.").

Thus, Applicants submit that the claims are patentable and the rejection of claims 1-7, 21-25, 27-31, and 33-37 should be withdrawn because each independent claim recites that "the selected pixels are **different** from the particular pixel" in combination with "determining a blurred value as a function of **only** the values of the selected pixels."

Regarding the rejection of claim 8 and claims 9-13 depending from claim 8, claim 8 has been amended to be in independent form. With regard to claim 8, the Examiner admits that Herf and Koike are deficient:

Koike and Herf do not disclose using these different sizes of filters or the number of selected pixels used to compute the blurred or smoothed value, but it is known to one of ordinary skill in the art that varying these values will affect the images in certain ways. It is known to one of ordinary skill in the art that the value P (the number of selected pixels), which is directly related to N, affects the blurred (or smoothed) value by averaging it over range of pixels. For example if

Applicants submit that the calculating and replacing as recited by claim 8 was not known within the context of pixel blurring at the time of the invention. Thus, Applicants traverse the Examiner's suggestion that such calculating within the context of pixel blurring would be common knowledge and request that the Examiner cite a reference to support the rejection or allow claims 8-13.

Thus, claims 1-13, 21-25, 27-31, and 33-37 are patentable and the rejection should be withdrawn. Claims 39-43 have been canceled.

6. Claims 1, 14, 15, 21, 26, 27, 32, 33, 38, 39 and 44 stand rejected as being unpatentable over Herf in view of Koike and Kawano (6480302). The Examiner cites Kawano to teach grayscale processing. However, Kawano is deficient for the same reasons as Herf and Koike. In particular, Kawano generally teaches processing of all pixels and does not address the recital that "the selected pixels are **different** from the particular pixel" in combination with "determining a blurred value as a function of **only** the values of the selected pixels."

Applicants submit that claims 1, 14, 15, 21, 26, 27, 32, 33, and 38 are patentable. Claims 39 and 44 have been canceled.

In view of the foregoing, favorable reconsideration and allowance of all claims is requested. The fact that Applicants may not have specifically traversed any particular assertion by the Office should not be construed as indicating Applicants' agreement therewith.

Applicants wish to expedite prosecution of this application. If the Examiner deems the application to not be in condition for allowance, the Examiner is invited and encouraged to telephone the undersigned to discuss making an Examiner's amendment to place the application in condition for allowance.

It is felt that a full and complete response has been made to the Office action and, as such, places the application in condition for allowance. Such allowance is hereby respectfully requested. If the Examiner feels, for any reason, that a personal interview will expedite the prosecution of this application, he is invited to telephone the undersigned.

Applicants do not believe that a fee is due in connection with this response. If, however, the Commissioner determines that a fee is due, he is authorized to charge Deposit Account No. 19-1345.

Respectfully submitted,

/Frank R. Agovino/

Frank R. Agovino, Reg. No. 27,416
SENNIGER POWERS
One Metropolitan Square, 16th Floor
St. Louis, Missouri 63102
(314) 231-5400

FRA/caa

Manual/Oversampling (Antialiasing)

From BlenderWiki

< Manual

Contents

Oversampling

Mode: All Modes

Panel: Render Context → Render

Hotkey: F10

Description

A computer generated image is made up of pixels, these pixels can of course only be a single colour. In the rendering process the rendering engine must therefore assign a single colour to each pixel on the basis of what object is shown in that pixel. This often leads to poor results, especially at sharp boundaries, or where thin lines are present, and it is particularly evident for oblique lines. To overcome this problem, which is known as *Aliasing*, it is possible to resort to an Anti-Aliasing technique. Basically, each pixel is 'oversampled', by rendering it as if it were 5 pixels or more, and assigning an 'average' colour to the rendered pixel. The buttons to control Anti-Aliasing, or OverSAmple (OSA), are below the rendering button in the *Render* Panel (*Render Panel.*).

Options

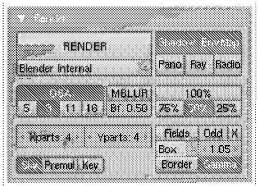
OSA
: Enables oversampling

5 / 8 / 11 / 16
: The number of samples to use. The values of *OSA* (5, 8, 11, 16) are pre-set numbers in specific sample patterns; a higher value produces better edges, but slows down the rendering. By default, we use in Blender a fixed "Distributed Jitter" table. The samples within a pixel are distributed and jittered in a way that guarantees two characteristics:

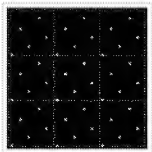
1. Each sample has equal distances to its neighbour samples

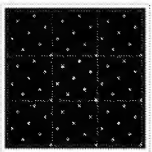
2. The samples cover all sub-pixel positions equally, both horizontally and vertically

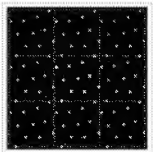
   The images below show Blender sample patterns for 5, 8, 11 and 16 samples. To show that the distribution is equalized over multiple pixels, the neighbour pixel patterns were drawn as well. Note that each pixel has an identical pattern.

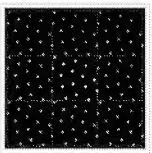


Render Panel.


5 samples


8 samples


11 samples


16 samples

Filtering

When the samples have been rendered, we've got color and alpha information available per sample. It then is important to define how much each sample contributes to a pixel. The simplest method is to average all samples and make that the pixel color. This is called using a "Box Filter". The disadvantage of this method is that it doesn't take into account that some samples are very close to the edge of a pixel, and therefore could influence the color of the neighbour pixel(s) as well.

Filter menu
: The filter type to use to 'average' the samples:

Box
: The original filter used in Blender, relatively low quality. For the Box Filter, you can see that only the samples within the pixel itself are added to the pixel's color. For the other filters, the formula ensures that a certain amount of the sample color gets distributed over the other pixels as well.

Tent
: A simplistic filter that gives sharp results

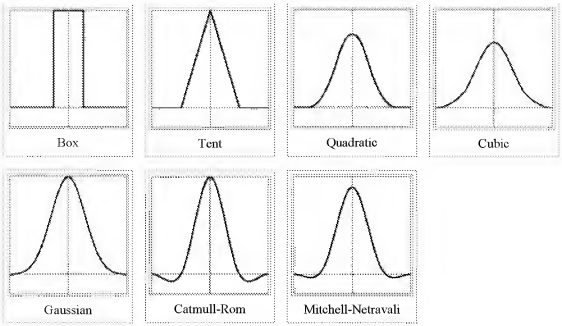
Quad
: A Quadratic curve

Cubic
: A Cubic curve

Gauss
: Gaussian distribution, the most blurry

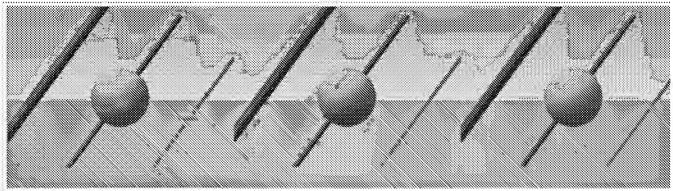
CatRom
: Catmull-Rom filter, gives the most sharpening

Mitch
: Mitchell-Netravali, a good allround filter that gives reasonable sharpness

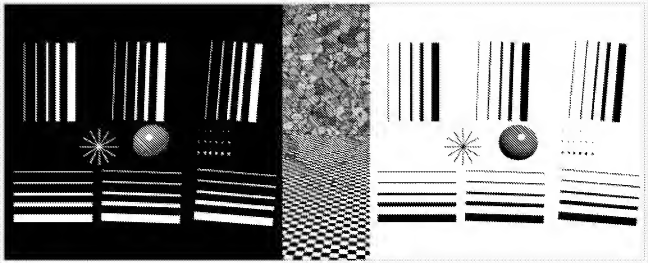


Making the filter size value smaller will squeeze the samples more into the center, and blur the image more. A larger filter size make the result sharper. Notice that the last two filters also have a negative part, this will give an extra sharpening result.

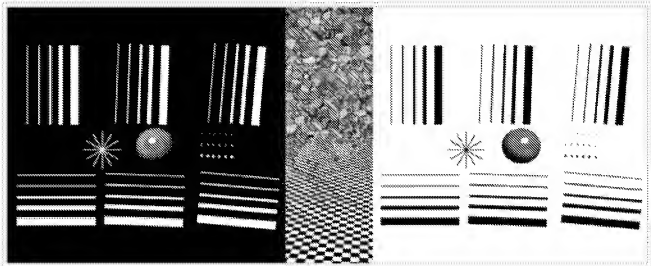
Examples



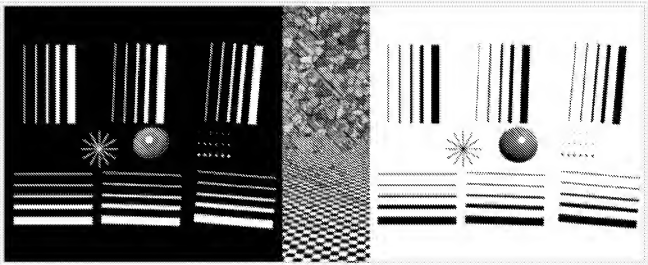
Rendering without OSA (left) with OSA=5 (center) and OSA=8 (right).



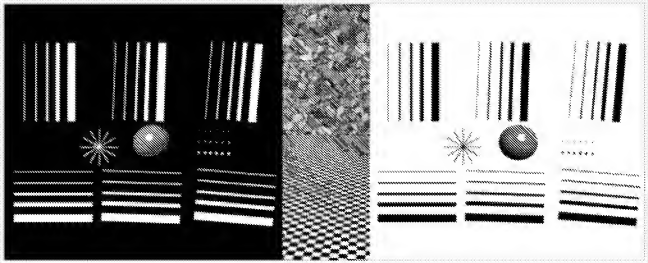
OSA 8, Box filter



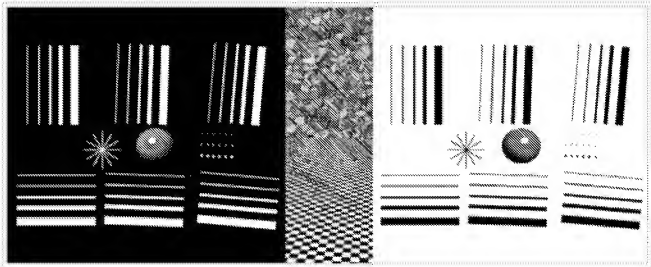
OSA 8, Tent filter



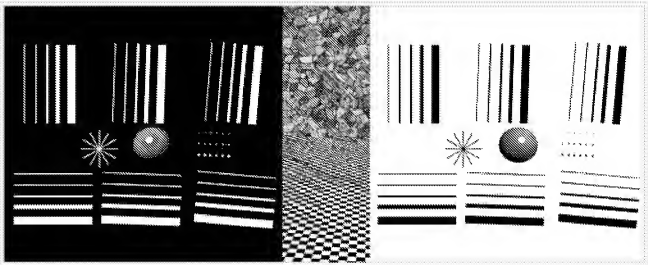
OSA 8, Quadratic filter



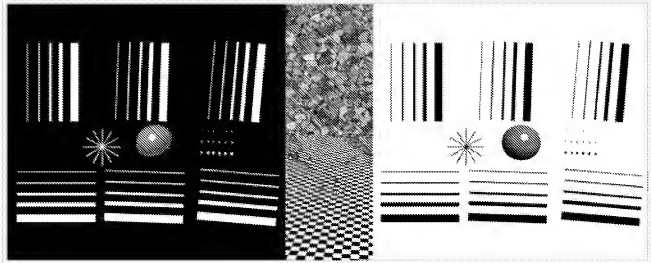
OSA 8, Cubic filter



OSA 8, Gaussian filter



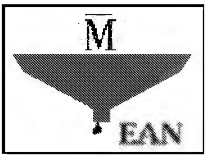
OSA 8, Catmull-Rom filter



OSA 8, Mitchell-Netravali filter

Previous: Manual/Rendering Contents *Next:* Manual/Rendering Options

Mean Filter



Common Names: Mean filtering, Smoothing, Averaging, Box filtering

Brief Description

Mean filtering is a simple, intuitive and easy to implement method of *smoothing* images, *i.e.* reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images.

How It Works

The idea of mean filtering is simply to replace each pixel value in an image with the mean ('average') value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the mean. Often a 3×3 square kernel is used, as shown in Figure 1, although larger kernels (*e.g.* 5×5 squares) can be used for more severe smoothing. (Note that a small kernel can be applied more than once in order to produce a similar but not identical effect as a single pass with a large kernel.)

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

Figure 1 3×3 averaging kernel often used in mean filtering

Computing the straightforward convolution of an image with this kernel carries out the mean filtering process.

Guidelines for Use

Mean filtering is most commonly used as a simple method for reducing noise in an image.

We illustrate the filter using

The image

shows the original corrupted by Gaussian noise with a mean of zero and a standard deviation (σ) of 8.

The image

shows the effect of applying a 3×3 mean filter. Note that the noise is less apparent, but the image has been `softened'. If we increase the size of the mean filter to 5×5, we obtain an image with less noise and less high frequency detail, as shown in

The same image more severely corrupted by Gaussian noise (with a mean of zero and a σ of 13) is shown in

The image

is the result of mean filtering with a 3×3 kernel.

An even more challenging task is provided by

. It shows an image containing `salt and pepper' shot noise.

The image

shows the effect of smoothing the noisy image with a 3×3 mean filter. Since the shot noise pixel values are often very different from the surrounding values, they tend to significantly distort the pixel average

calculated by the mean filter.

Using a 5×5 filter instead gives

This result is not a significant improvement in noise reduction and, furthermore, the image is now very blurred.

These examples illustrate the two main problems with mean filtering, which are:

- A single pixel with a very unrepresentative value can significantly affect the mean value of all the pixels in its neighborhood.
- When the filter neighborhood straddles an edge, the filter will interpolate new values for pixels on the edge and so will blur that edge. This may be a problem if sharp edges are required in the output.

Both of these problems are tackled by the median filter, which is often a better filter for reducing noise than the mean filter, but it takes longer to compute.

In general the mean filter acts as a lowpass frequency filter and, therefore, reduces the spatial intensity derivatives present in the image. We have already seen this effect as a `softening' of the facial features in the above example. Now consider the image

which depicts a scene containing a wider range of different spatial frequencies. After smoothing once with a 3×3 mean filter we obtain

Notice that the low spatial frequency information in the background has not been affected significantly by filtering, but the (once crisp) edges of the foreground subject have been appreciably smoothed. After filtering with a 7×7 filter, we obtain an even more dramatic illustration of this phenomenon in

Compare this result to that obtained by passing a 3×3 filter over the original image three times in

Common Variants

Variations on the mean smoothing filter discussed here include *Threshold Averaging* wherein smoothing is applied subject to the condition that the center pixel value is changed only if the difference between its original value and the average value is greater than a preset threshold. This has the effect that noise is smoothed with a less dramatic loss in image detail.

Other convolution filters that do not calculate the mean of a neighborhood are also often used for smoothing. One of the most common of these is the Gaussian smoothing filter.

Interactive Experimentation

You can interactively experiment with this operator by clicking here.

Exercises

1. The mean filter is computed using a convolution. Can you think of any ways in which the special properties of the mean filter kernel can be used to speed up the convolution? What is the *computational complexity* of this faster convolution?

2. Use an edge detector on the image

and note the strength of the output. Then apply a 3×3 mean filter to the original image and run the edge detector again. Comment on the difference. What happens if a 5×5 or a 7×7 filter is used?

3. Applying a 3×3 mean filter twice does not produce quite the same result as applying a 5×5 mean filter once. However, a 5×5 convolution kernel *can* be constructed which is equivalent. What does this kernel look like?

4. Create a 7×7 convolution kernel which has an equivalent effect to three passes with a 3×3 mean filter.

5. How do you think the mean filter would cope with Gaussian noise which was not symmetric about zero? Try some examples.

References

R. Boyle and R. Thomas *Computer Vision: A First Course*, Blackwell Scientific Publications, 1988, pp 32 - 34.

E. Davies *Machine Vision: Theory, Algorithms and Practicalities*, Academic Press, 1990, Chap. 3.

D. Vernon *Machine Vision*, Prentice-Hall, 1991, Chap. 4.

Local Information

Specific information about this operator may be found here.

More general advice about the local HIPR installation is available in the *Local Information* introductory section.

computer graphics

Fall 2005

Image Processing

To implement some of the requirements of this project, you will need to know a little about image processing. More specifically, this document will cover some of the aspects of edge detection and image gradients.

Some Basics

Often in image processing, we apply an operation to just a small region of an image at a time. One of the most common techniques used for applying such operations is by using something called a filter. Diffenent image filters are distinguished by their *kernel* and their *support*. Basically, you can think of a filter kernel as a grid of values that we overlay on top of the pixels of an image. Here is an example of a kernel:

0	1	1	1	0
1	1	2	1	1
1	2	3	2	1
1	1	2	1	1
0	1	1	1	0

We apply an operation to a pixel in the image as follows. First, imagine placing the center of the kernel on top of the pixel of interest. Multiply the value at each position in the kernel by the color of the pixel underneath that position. Then, add up all these products, and divide by the sum of the values in the kernel (if it's nonzero).

We can do many interesting things by changing the values in the kernel. The rest of this section will explain how kernels and other image manipulation methods are used to find edges and gradients in an image.

Image Blurring

The first step in finding edges and gradients in an image is to blur the image. (Note: for finding edges

and image gradients, we use a grayscale version of the original image.) Blurring reduces the "noise" (randomness) in the picture, and allows us to find edges and determine image gradients more accurately.

To get a grayscale version of the image, we take the average of the red, green, and blue channels. However we use the following weights:

Intensity = 0.299R + 0.587G + 0.114B

Why do we use these weights? Well, the human eye is better at detecting some colors than others. It picks up green the best, followed by red then blue. We're interested in producing an image that represents what the eye thinks the intensity of light is at each pixel, and this weighting achieves that effect.

There are several choices for the filter kernel to use for blurring image. We will present three of them here.

- Box Filter: The first, and the simplest, is the "box filter". Basically, the kernel for the box filter is filled with all 1s.
- Bartlett Filter: Somewhat more complicated is the Bartlett filter. In this, pixels closer to the center are weighted more heavily than those further away. For the Bartlett filter, imagine placing a cone on top of the kernel, and using the height of the cone at each location to determine that position's relative weight.
- Gaussian Filter: Like the Bartlett filter, pixels near the center are weighted more heavily in the Gaussian filter. However, instead of a cone placed over the kernel, imagine a 3-dimensional bell curve on top of it.

In general, which filter you should use depends on the image you're manipulating and what you want to accomplish. The box filter is the easiest to implement, but the Bartlett and Gaussian filters often produce "better" blurring results.

Edge Detection

Once you have blurred the grayscale version of the original image, you are ready to do edge detection. The method of edge detection we present here is known as Sobel edge detection.

To do Sobel edge detection, we once again use filters. This time we make two passes over the image, using a different kernel each time. Here are the filter kernels that are used.

1	2	1
0	0	0
-1	-2	-1

1	0	-1
2	0	-2

1	0	-1

By using both these filter kernels on each pixel in the image, we get two new values at each pixel location (call them temp1 and temp2). We then produce a single value at each location using this formula: sqrt(temp1*temp1 + temp2*temp2). (You may notice a striking similarity to the distance formula used in mathematics.) Finally, we say that if this single value is above a certain threshold (usually specified by the user), then there is an edge at that location.

Image Gradients

Given a blurred image, we can also produce the gradients for an image. To intuitively understand what the image gradient is, recall what a gradient in is in mathematics --- the direction of maximum rate of change. For a surface (such as an array of pixels), the gradient is the direction in which the color changes the fastest (and perpendicular to the gradient, the color changes the slowest --- i.e., it's close to the same color).

Why is this important to us (and important for the Impressionist program, in particular)? Well, if we know (at each pixel) the direction in which the color is changing the least, we can orient our brush strokes in that direction to make it more realistic. (Because if we paint a stroke in a certain direction, the color of the paint stays the same during that stroke!)

So how do we implement this? If you answer contained the word "filters," you're on the right track. We will actually produce two gradients, the x-gradient (telling how fast the color changes along each row), and the y-gradient (telling how fast the color changes along each column). Here are the kernels that are used.

x-gradient filter kernel:

-1	1

y-gradient filter kernel:

1
-1

Using these kernels gives us two separate images, one of which tells us how fast the color changes (at each pixel) in the x-direction, and one telling us how fast it changes in the y-direction. Finally, we use this information to determine the direction of the maximum rate of change in color, and we orient our brush strokes perpendicular to that direction.